# Network Load Monitoring in Distributed Systems

Kazi M Jahirul Islam[*], Behrooz A. Shirazi[*], Lonnie R. Welch[+], Brett C. Tjaden+, Charles Cavanaugh[*], Shafqat Anwar[*]

[*]University of Texas at Arlington Department of CSE Box 19015 Arlington, TX 76019-0015

{islam|shirazi|cavan}@cse.uta.edu, anwar@swbell.net

[+]Ohio University School of Electrical Engineering and Computer Science Athens, OH 45701-2979

{welch|tjaden}@ohio.edu

**Abstract.** Monitoring the performance of a network by which a real-time distributed system is connected is very important. If the system is adaptive or dynamic, the resource manager can use this information to create or use new processes. We may be interested to determine how much load a host is placing on the network, or what the network load index is. In this paper, a simple technique for evaluating the current load of network is proposed. If a computer is connected to several networks, then we can get the load index of that host for each network. We can also measure the load index of the network applied by all the hosts. The dynamic resource manager of DeSiDeRaTa should use this technique to achieve its requirements. We have verified the technique with two benchmarks – LoadSim and DynBench.

## 1 Introduction

The DeSiDeRaTa project is providing innovative resource management technology that incorporates knowledge of resource demands in the distributed, real-time computer control systems domain. This project involves building middleware services for the next generation of ship-board air defense systems being developed by the U.S. Navy. DeSiDeRaTa technology differs from related work in its incorporation of novel features of dynamic real-time systems. The specification language, mathematical model and dynamic resource management middleware support the dynamic path paradigm, which has evolved from studying distributed, real-time application systems. The dynamic path is a convenient abstraction for expressing end-to-end system objectives, and for analyzing timeliness, dependability and scalability. Novel aspects of the dynamic path paradigm include its large granularity and its ability to accommodate systems that have dynamic variability[1].

The resource manager is responsible for making all resource allocation decisions. The resource manager component computes allocation decisions by interacting with the system data repository and obtaining software and hardware system profiles. The allocation decision may involve migrating programs to different hosts, starting additional copies of programs (for scalability), or restarting failed programs (for survivability). The system data repository component is responsible for collecting and maintaining all system information.

The resource management architecture (attached at end of this document) consists of components for adaptive resource management and QoS negotiation, data broker, path monitoring and diagnosis, resource monitoring, and resource management consoles. The adaptive resource management and QoS negotiation component is responsible for making resource management decisions. This component computes the allocation decision by interacting with the data broker and obtaining software and hardware system profiles. The allocation decision may involve migrating programs to different host nodes, starting additional copies of programs (for scalability), or restarting failed programs (for survivability). The resource management component carries out its decisions by communicating with a daemon program (on each host) to start up and control programs on each host[4].

The data broker component is responsible for collecting and maintaining all system information. The data broker reads the system description and requirements expressed using the specification language and builds the data structures that model the system. Dynamically measured software performance metrics, such as path latency and throughput, and resource usage characteristics, such as program page faults and resident size, are collected and maintained by the path monitoring and diagnosis component. The data broker obtains measurements of the dynamic attributes of the software from the monitoring component. Hardware resource profiles are collected and maintained by the resource monitoring component, and fed to the data broker on demand as well as periodically. The data broker thus provides a single interface for all system data. The path monitoring and diagnosis component monitors the performance of software systems at the path-level. This component determines the changing requirements of the software by interacting with the data broker. When a path fails to meet the requirements, this component performs diagnosis of the path, and determines the ``bottleneck'' node of the path. Resource management consoles display system and path status, and allow dynamic attributes, such as deadlines, to be modified. All communication for such consoles is through the data broker[6].

As mentioned earlier, the resource manager utilizes software and hardware system profiles to make allocation decisions. To obtain the system profiles, the resource manager continuously monitors the whole system and calculates various metrics. These metrics provide the guidelines for choosing among different allocation possibilities and optimizing resource usage. One of the components that is monitored by the resource manager is the network which connects the different computers that form the distributed real-time system. Several network parameters are of interest including host-to-host delay, network load index, host load index, etc. Host-to-host delay measures the time required to transmit a message from a specific host to another. Host load index measures the load applied to the network by a specific host. Network load index measures the total amount of load applied to the network by all the hosts that are currently connected or communicating through the network. Furthermore, since computers may be connected to multiple networks, they may have multiple IP addresses. Therefore, we can measure the host load index of a host on a specific network; or we may be interested in the load index of each network[7].

Depending on these parameters, the resource manager might select a different host to initiate a new process; or it might send data to a different host to get the result in an acceptable time using the least busy network or route. In a multi-homed network, it might also route data through an alternate network where load index indicates low traffic.

In this paper, we will formulate a host load index and network load index. We will also explain the experimental procedure that was followed to get the results. At the end we will also discuss the limitations of our approach and present some ideas for future work that will allow us to improve our method.

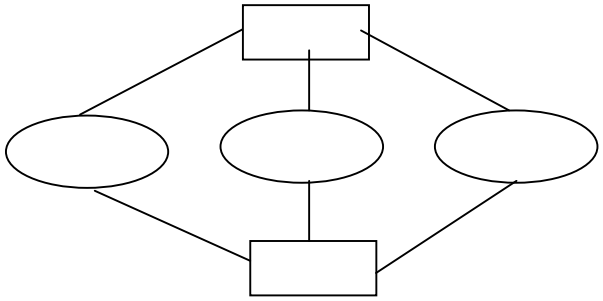

**Fig. 1.** Formal definition of the problem

We will use the above model for the illustration of the problem. Let us assume that there are n hosts $Host_1$, $Host_2$, … $Host_n$ and m networks $Net_1$, $Net_2$, …$Net_m$ in the system (Figure 1). We are interested in finding the host load index for each host, $Host_i$. We are also interested in finding the load index on all the networks through which the hosts are connected. That means, if they are connected through k different networks, we are interested in measuring the load index on all k different networks. We also want to measure the load index of each network $Net_i$. That will help us to select the least loaded network for transmission.

## 2 Load Simulator

The LoadSimulation[1], hereafter referred as LoadSim, is able to compose and simulate the resource utilization (CPU cycles, network bandwidth, latency, etc.) of a large-scale distributed system that may consists of many interacting processes executing on many computers networked together. Here, simulation of distributed system load is achieved by means of replicated copies of a configurable LoadSim computer program. Each replicated copy must be capable of being initialized to a potentially different host computer and network resource utilization profile. LoadSim replicates are mapped onto a heterogeneous network of computers by a set of support services that allows the user to specify and control the topology and characteristics of the LoadSim configuration under test. The LoadSim further provides the ability to collect metrics on the performance of the simulated large-scale system.

---

[1] Some parts of LoadSimulation have been taken from "Requirements for a Real-time Distributed LoadSimulation" written by Timothy S. Drake. He may be contacted at drakets@nswc.navy.mil

The primary goals of the benchmark are to provide the ability to objectively assess the network communication protocols (e.g. TCP/IP, UDP, etc.), network bandwidth, network latency characteristics and to place additional load on partial implementations of real systems in order to assess the impact of the load that would be placed on the computing resource base by missing components if those components were present.

As LoadSim can place additional load on a partial implementations of real systems, we used this tool to apply load on the network. The DynBench benchmark application is modeled after typical distributed real-time military applications such as an air defense subsystem. Figure 2 shows the three *dynamic paths* from the DynBench benchmark application. The *detect* path (path 1) is a continuous path that performs the role of examining radar sensor data (radar tracks) and detecting potential threats to a defended entity. The sensor data are filtered by software and are passed to two evaluation components, one is software and the other is a human operator. The detection may be performed manually, automatically, or semi-automatically (automatic detection with manual approval of engagement recommendation). When a threat is detected and confirmed, the transient *engage* path (path 2) is activated, resulting in the firing of a missile to engage the threat. After a missile is in flight, the quasi-continuous *guidance* path (path 3) uses sensor data to track the threat, and issues guidance commands to the missile. The *guidance* path involves sensor hardware, software for filtering/sensing, software for evaluating and deciding, software for acting, and actuator hardware.[5]

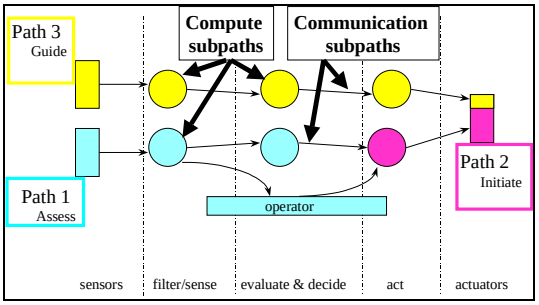


**Fig. 2.** The DynBench dynamic paths

A (simulated) radar sensor periodically generates a stream of data samples (representing the positions of moving bodies) based on equations of motion defined in a *scenario* file. The data stream is provided to the Filter Manager, which distributes the current workload among replicas of the filter program (Figure 3). Each filter uses a least mean square regression algorithm to filter “noise” and to correlate the data points into three equations that describe the motion of a body. The equations of motion for each of the observed bodies are sent to the evaluate and decide manager, which distributes the workload among the evaluate and decide programs. Evaluate and decide processes determine if the current position of an observed body is within a “critical region” defined by a doctrine file.

When a body first enters the critical region, it is passed to the action manager in the initiation path. Action manager distributes the workload among the action programs, which calculate equations of motion to intercept bodies of interest. A simulated actuator operates the initiation of motion of the intercepting body.

Whenever engaged objects are present in the sensor data, the evaluate and decide programs transmit the equations of motion of those bodies to the monitor and guide manager, which pairs identified bodies of targets with their corresponding interceptors. The corresponding pairs of equations are equally distributed among monitor and guide processes, which monitor the progress of the interceptor relative to the new positions of their intended target. If necessary, a new fight equation is calculated for the interceptor and sends to sensor. If an interception occurs, this process sends a request to remove the target and interceptor from the data stream.

There is also a deconflict path added to DynBench application subsystem. The deconflict path is designed to pre-launch the intercept bodies, and check if there is some conflict for the interceptor flight path with some other tracks or interceptors before the interceptor hit its target track. If there is a conflict, deconflict will send a warning message to Radar Display.

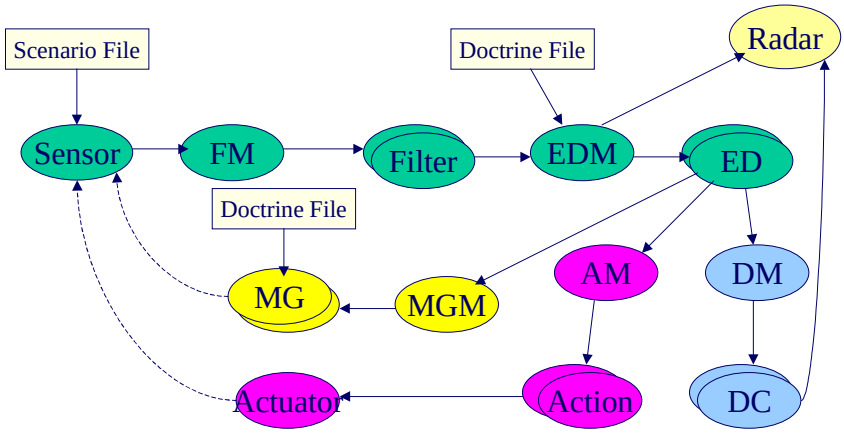


**Fig. 3.** DynBench application subsystem

We have used the DynBench as the load generator. We can put specific amount of tracks in the sensor and that will be processed by the whole benchmark. The increasing amount of tracks will put increasing amount of load on the network.

## 3 Previous Work

Philip M. Irey IV, Robert D Harrison and David T. Marlow have previously analyzed LAN performance in [2] and have shown an approach to evaluate the applicability of currently available commercial products in real-time distributed systems. They also had defined a few parameters for this purpose. Depending on these parameters and

explained measurement methodologies we can determine the applicability of the commercial products in real-time systems. Andrej Sostaric, Milan Gabor Andreas Gygi had developed Mtool[3]. It can be used in performance monitoring in networked multi-platform systems. They emphasized on the three-tier architecture and used Java technology to achieve platform independence.

## 4 Experimental Procedure

We used *netstat* to collect our statistics. We are mainly interested in the TCP/IP suite of protocols, and the command '*netstat –i 1*' is used to produce a packet transmission summary once per second. We used this command in different hosts that may be connected through one or more networks to gather the statistics. This command gives number of packets in, number of packets out and number of collision in every second. We tried to formulate the load index using any linear combination of these three parameters.

To generate the load, we used two different tools. One is load simulator. We put the specific amount of load in every 200 millisecond on the network. The other tool is DynBench, a benchmark for DeSiDeRaTa. We were sending a specific amount of tracks or load from Sensor to Filter Manager for a specific amount of time. We tested for each load for approximately three minutes; that means each amount of load was applied on the network for three minutes. Later, we increased the load and run the experiment again. The test continued in this manner.

We performed our experiments in Sun workstations that are connected to a LAN. We collected statistics from different hosts in both cases. We are sure that the load was sent from the specified source host to the specific destination host. Because the mean and standard deviation of packets in, packets out and collision of the source and destination differs significantly from the other hosts; both mean and standard deviation were much higher in the source and destination.

| Load generator | DynBench | | | Load Simulator | | |
|---|---|---|---|---|---|---|
| | Host 1 | Host 2 | Network | Host 1 | Host 2 | Network |
| Out | 0.53306 | 0.62177 | 0.57686 | 0.90124 | 0.92911 | 0.91584 |
| Out+Collision | 0.54933 | 0.62177 | 0.58216 | 0.94287 | 0.92911 | 0.94481 |
| Out+in | 0.53596 | 0.61334 | 0.58968 | 0.91077 | 0.91070 | 0.91472 |
| In | 0.53698 | 0.60707 | 0.59190 | 0.92778 | 0.90018 | 0.91252 |
| Collision | 0.55735 | N/A | 0.56566 | 0.96815 | N/A | 0.97252 |
| In+Collision | 0.55380 | 0.60707 | 0.60402 | 0.96350 | 0.90018 | 0.94345 |
| In+Out+Collisn. | 0.54741 | 0.61334 | 0.59450 | 0.93961 | 0.91070 | 0.93282 |

**Table 1.** Coefficient of correlation

Table 1 shows the coefficient of correlation of different combination of the parameters with the load that was applied on the network. Here data is moving from "Host 1" to "Host 2". Two scenarios of load generation are shown here for comparison: the first load was produced by DynBench; the second load was generated from Load Simulator. Figs. 4 and 5 summarize the results. We have also measured the load index of the whole network using the same approach. In the above graph,

network specifies the load index of the whole network. It is the sum of the specified parameters.

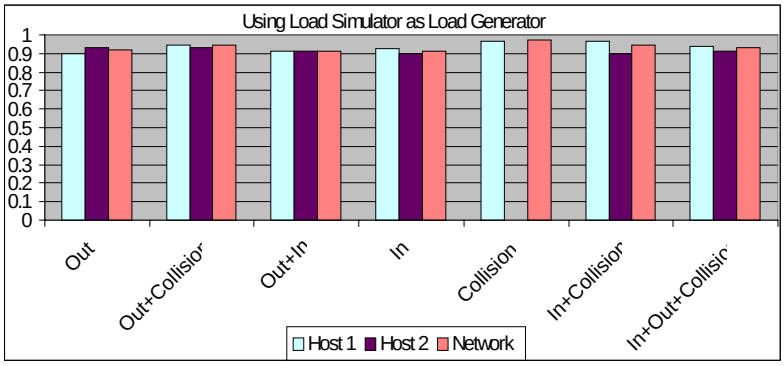


**Fig. 4.**

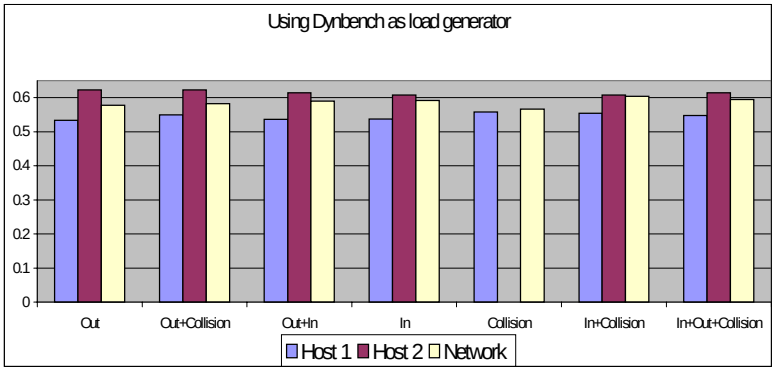


**Fig. 5.**

Several interesting characteristics can be observed from here. When a host is transmitting data to another host, the number of collisions gives a better approximation of the load index at the source than at the destination. From the receiver's point of view, the number of packets transmitted by the receiver reflects its load index. Here, both "out" and "out+collision" are the same. Experimental data shows that during packet transmission, the receiver does not experience any collision. As TCP/IP uses handshaking to inform the sender about the reception of data, the number of packets out somehow reflects actually about the number of packets transmitted to this host. This is supported by both load simulator and DynBench. For measuring the network load index, two different parameters are chosen by two

different load generators. DynBench proposes sum of “In+Collision” of all the hosts connected to the network while according to Load Simulator, it is sum of Collision.

## 5 Conclusion

In this paper, we have proposed a simple non-intrusive technique for measuring the load applied on a network. We have used a simple tool, *netstat –i*, for that. We generated load using DynBench and LoadSimulator and then measured the load index. We have shown all the combinations of three parameters – packet in, packet out and collision, to determine which one best describes the load index.

Another way of measuring load index is to send time-stamped packets from one host to another host. This can measure the delay in the network, which also gives a fair indication of applied load. We can also measure the delay between two hosts through each network they are connected to. This gives the network load index.

## References


1. L. R. Welch, B. A. Shirazi, B. Ravindran and C. Bruggeman, “DeSiDeRaTa: QoS Management Technology for Dynamic, Scalable, Dependable, Real-Time Systems”, Proceedings of The 15th IFAC Workshop on Distributed Computer Control Systems, September 1998.
2. Philip M. Irey IV, Robert D Harrison and David T. Marlow, “Techniques for LAN Performance Analysis in a Real-Time Environment”, Real-Time Systems, 14 21-44(1998) Kluwer Academic Publishers.
3. Andrej Sostaric, Milan Gabor Andreas Gygi, “Performance Monitoring in Network Systems”, 20th Int. Conf. Information Technology Interfaces ITI ’98, June 16-19, 1998.
4. L. R. Welch, B. Ravindran, B. Shirazi, and C. Bruggeman, “Specification and analysis of dynamic, distributed real-time systems”, in Proceedings of the 19th IEEE Real-Time Systems Symposium, 72-81, IEEE Computer Society Press, 1998
5. L. R. Welch, B. A. Shirazi, "A Dynamic Real-Time Benchmark for Assessment of QoS and Resource Management Technology", RTAS 99
6. B. Ravindran, L. R. Welch, B. A. Shirazi, Carl Bruggeman, Charles Cavanaugh, "A Resource Management Model for Dynamic, Scalable, Dependable, Real - Time Systems"
7. L. R. Welch, P. Shirolkar, Shafqat Anwar, Terry Sergeant, B. A. Shirazi, "Adaptive Resource Management for Scalable, Dependable, Real - Time Systems"